MARKER ARGUMENTATION FOR AN INTEGRATED CIRCUIT DESIGN TOOL AND FILE STRUCTURE

TECHNICAL FIELD

[0001] The present invention relates to electronic circuit fabrication, and more particularly to a computer-based method and architecture that automatically transforms a behavioral description of a circuit into a hardware description of the circuit and inserts markers to simplify the subsequent processing used to prove equivalence of the behavioral and hardware descriptions.

BACKGROUND OF THE INVENTION

[0002] In order to fabricate an electronic circuit such as an application specific integrated circuit (ASIC) or a field programmable gate array (FPGA), a number of processes must be performed. For example, the first step is to design the circuit at a "behavioral" level. The behavioral description is the top level of the design hierarchy. A hardware description characterizes the circuit using actual hardware components and is the lowest level of the design hierarchy.

[0003] The behavioral description is typically a block diagram or netlist that describes the various functions of the circuit in relatively general terms. The hardware description is based on the behavioral description. The behavioral description does not have enough specificity to fully enable a circuit manufacturer to implement the circuit design. The conventional approach is to manually generate an electronic file that characterizes the circuit at the hardware level in a programming language such as hardware description language (HDL). A design engineer goes through the behavioral

description on a component-by-component basis and generates the necessary code sets in the programming language. This approach is labor intensive and subject to human error.

[0004] Once the electronic file has been manually generated, a complex verification process is required. In the case of the ASIC design, the functionality is extensively verified from a model of the circuit written in C-code. A select set of C-code input simulation vectors (i.e. input sequences) are then also applied to the HDL circuit model. In this co-simulation, the output vectors from the HDL model are compared bit by bit to the corresponding output vectors from the C-code model to establish a correspondence between the performance verified by the C-code model and the hardware actually being fabricated. This is typically required due to the complexity of the ASIC chip and the difficulty in performing a functional simulation in HDL, the complexity and cost of the ASIC fabrication process, and the need for a high level of certainty as to the functionality and first pass success of the electronic file and its subsequent fabrication. The manual generation of the C model also requires detailed knowledge of the C programming language, which increases the labor costs that are associated with fabrication. Once the electronic file has been verified, it is typically converted to the format used by the manufacturer using a conversion program such as Synopsis. After the electronic file has been converted, the manufacturer fabricates the circuit.

[0005] In U.S. Patent No. 6,077,303 to Mandell et al. and my co-pending application "Architectural Structure Of A Process Netlist Design Tool", Serial No. _____, filed _____ a design tool and method for simplifying the design of

ASICs is disclosed. The design tool and method generates symbolic and numeric equations. In situations involving relatively less complex, small to medium ASICs, the symbolic equations that are generated by the design tool are finite and computationally feasible. In other words, the symbolic equations require a limited amount of computer storage and running time to be further simplified. However, when more complex circuits are involved (such as large ASICs and/or circuits including cascaded finite impulse response (FIR) filters, closed-loops, and other complex components), the resulting set of symbolic equations become large enough to overflow the memory that is available to the workstation and/or the capability of the symbolic manipulation tool that is used to compare the equations.

SUMMARY OF THE INVENTION

[0006] A design tool and method according to the invention characterizes a circuit at a hardware level description. A behavioral level description of the circuit is created. Symbolic equations for components of the behavioral level description are created. The behavioral level description is partitioned by inserting a marker component between first and second components of the circuit to simplify the subsequent processing used to prove the equivalence of the behavioral and hardware level descriptions.

[0007] In other features of the invention, the symbolic equations are back-substituted until output variables are expressed in terms of input variables that determine the output variables. The marker component is defined using a unique

symbolic name. Current time counts of each clock cycle are used to compute an index for the marker component.

[0008] It is to be understood that both the foregoing general description and the following detailed description are merely exemplary of the invention, and are intended to provide an overview or framework for understanding the nature and character of the invention as it is claimed. The accompanying drawings are included to provide a further understanding of the invention, and are incorporated in and constitute part of this specification. The drawings illustrate various features and embodiments of the invention, and together with the description serve to explain the principles and operation of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present invention will become more fully understood from the detailed description and the accompanying drawings, wherein:

[0010] FIG. 1 is a flowchart of a computer-implemented method for characterizing a circuit at a hardware level;

[0011] FIG. 2 illustrates a method for implementing modifications with a set of predefined modules;

[0012] FIG. 3A is a block diagram of an original architecture of a circuit before application of an architecture sharing command;

[0013] FIG. 3B is a block diagram of a reduced architecture resulting after application of the architecture sharing command;

[0014] FIG. 4 is a block diagram illustrating the reduction of control logic;

[0015] FIG. 5A is a block diagram of a single path in accordance with the principles of the present invention;

[0016] FIG. 5B is a block diagram of a set of parallel paths in accordance with principles of the present invention;

[0017] FIG. 6 is a block diagram showing a two-way multiplex control counter;

[0018] FIG. 7A is a block diagram of an original component before application of an implementation assignment command;

[0019] FIG. 7B is a block diagram illustrating implementation components after application of an implementation assignment command;

[0020] FIG. 8 is a block diagram illustrating delay shifting;

[0021] FIG. 9 is a block diagram of an exemplary circuit topology;

[0022] FIG. 10 is a block diagram illustrating the exemplary circuit topology of FIG. 9 at time t=1;

[0023] FIG. 11 is a table illustrating string values and outputs printed to an equation file;

[0024] FIG. 12 illustrates an exemplary circuit topology with equations that grow quickly and are likely to cause processing overload;

[0025] FIG. 13 illustrates a circuit topology with marker components;

[0026] FIG. 14 illustrates high level functional requirements of the marker component;

[0027] FIG. 15 illustrates steps for inserting the marker component;

[0028] FIG. 16 illustrates additional high level functional requirements for the marker component;

[0029] FIG. 17 illustrates a marker component inserted in the exemplary circuit topology of FIG. 9;

[0030] FIG. 18 is a table illustrating string values and outputs printed to an equation file for the circuit topology of FIG. 17;

[0031] FIGs. 19 and 20 illustrate an output variable naming convention; and

[0032] FIG. 21 illustrates a generalized rate change component.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0033] The following description of the preferred embodiment(s) is merely exemplary in nature and is in no way intended to limit the invention, its application, or uses.

[0034] Turning now to FIG. 1, a computer-implemented design tool and method 20 for characterizing a circuit (not shown) at a hardware level is shown. The circuit is preferably an application specific integrated circuit (ASIC), but other types of circuits such as field programmable gate arrays (FPGA) can also benefit from the present invention. Furthermore, the particular application for the circuit not critical.

[0035] Generally, the method 20 includes an initialization step 22 for receiving a circuit description 21 that characterizes the circuit at a behavioral level of a design hierarchy. The description is preferably in the form of a block diagram, a descriptive netlist or other suitable format. A netlist is a list of names of symbols or circuit elements and their inter-connections. Netlists can be captured from a schematic of a computer aided engineering (CAE) application.

[0036] Upon receiving a new computer-based command at step 24 (from a source such as a keyboard or a script driver input), the new command is interpreted at step 28. The new command requests a desired modification in the description of the circuit. For example, the modification may replace a digital filter with a delay chain and multipliers. At step 30, an electronic file is generated based on the new command such that the electronic file characterizes the circuit at the hardware level in a desired programming language. The method 20 is an iterative method in which the electronic file is "built-up" as commands are received. The hardware level is located at a lower level in the hierarchy (typically at the bottom level) than the behavioral level.

[0037] Although the desired programming language is preferably the hardware description language (HDL), it can be appreciated that the present invention is not so limited. In the case of an ASIC, a verification mechanism is preferably provided in the form of a C language file. Simulating the operation of the circuit with the C language file provides a mechanism for cross-checking the HDL file. It is important to note, however, that the present invention provides confirmation algorithms that (depending on the confidence requirements of the application) might obviate the need for the C language file.

[0038] Turning now to FIG. 2, the presently preferred approach to the step 28 of interpreting the new command may include one or more steps. For example, when a replacement command is received at step 40, an original component (e.g., digital filter) of the circuit residing at an original level in the hierarchy is replaced at step 38 with one or more replacement components (e.g., delay chain and multipliers) residing at a lower level in the hierarchy. Step 38 also confirms that the replacement components are

functionally equivalent to the original component. Detailed confirmation algorithms is discussed in greater detail below. At step 26, the hierarchical netlist is transformed (or updated) in accordance with the modifications.

[0039] An important aspect of ASIC design is the concept of architecture sharing. Architecture sharing involves the sharing of physical processing paths by symbolic (logical) processing in order to conserve resources. For example, a block diagram of a circuit at a behavioral level includes ten multipliers operating at 1 MHz. Architecture sharing permits the substitution of one multiplier operating at 10 MHz. When the architecture sharing command has been entered in step 32, an original architecture of the circuit is reduced at step 42 to provide a reduced architecture Furthermore, step 42 also confirms that the reduced architecture is functionally equivalent to the original architecture.

[0040] The process of confirming that the reduced architecture is functionally equivalent to the original architecture will now be described in greater detail. With continuing reference to FIGS. 3A and 3B, the original architecture 34 is described for an arbitrary n as follows:

$$X(n) = F[x(n)]; \quad X(n-1) = F[x(n-1)]$$
$$Y(n) = F[y(n)]; \quad Y(n-1) = F[y(n-1)]$$
$$Z(n) = F[z(n)]; \quad Z(n-1) = F[z(n-1)].$$

If the reduced architecture 36 is defined as

$$A(n) = \begin{cases} X\left(\frac{n}{3}\right) & \text{When n mod 3 = 0} \\ Y\left(\frac{n-1}{3}\right) & \text{When n mod 3 = 1} \\ Z\left(\frac{n-2}{3}\right) & \text{When n mod 3 = 2} \end{cases}$$

Then

$$C(n) = F[A(3n-3)] = X(n-1)$$
$$D(n) = F[A(3n-2)] = Y(n-1)$$
$$E(n) = F[A(3n-1)] = Z(n-1)$$

Thus, the original architecture 34 is equivalent to the reduced architecture 36. Specifically, it can be seen that interpolators 44 (or up converters) and decimators 46 (or down converters) permit the use of an instantaneous function F to be reduced by a factor of 3.

[0041] Returning now to FIG. 2, the present invention also provides for control logic reduction. This feature is particularly important in cases where architecture sharing results in an increase in control logic. When it is determined that a control logic reduction command has been received at step 48, redundant control logic of the circuit is eliminated and necessary control logic is modified at step 50 to produce a reduced architecture.

[0042] FIG. 4 illustrates the transformation of a group of logic circuits 52 into a rate change component (or reduced logic architecture) 54. It will be appreciated that the logic circuits 52 contain various rate change components such as delays, multiplexers, interpolators and decimators. It will further be appreciated that the logic circuits 52 can be defined as a set of path lists that interconnect a set of inputs to a set of outputs. FIGS. 5A and 5B demonstrate a single path 56 and a set of paths 58, respectively. The single path 56 between points A and B is therefore characterized by,

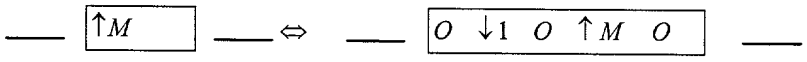
$$\boxed{K_L \downarrow N \quad K_m \uparrow M \quad K_R},$$

corresponding to the single path 56 of FIG. 5A.

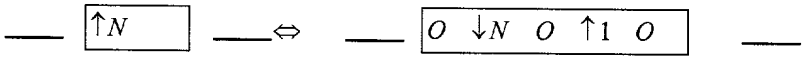
[0043] A path list is therefore made up of a set of paths 58 from point A to point B such that each path has the same $\downarrow N \; and \uparrow M$. It will be appreciated that in order to ensure that two outputs do not arrive concurrently, the set of paths 58 is constrained by $(Kri - KRj)\, MOD\, M \neq O \; for \; any \; i \neq j$. A rate change component 54 is defined as a set of path lists that interconnect the set of inputs as shown in FIG. 4.

[0044] The various multi-rate components can therefore be represented as

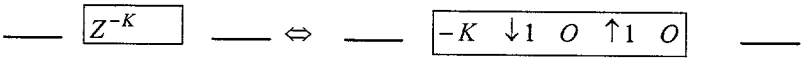
interpolation



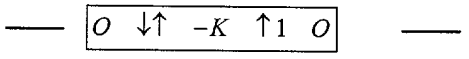
decimation



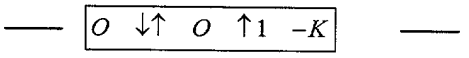
delay



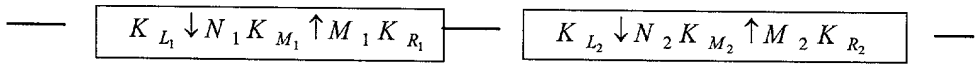
or

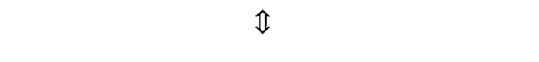


or



Furthermore, FIG. 6 illustrates that a 2-way counter-controlled mux 60 can be given by the counter representation 62. In order to resolve two paths in series we have





or

Specifically, the algorithm is given by

Algorithm:

(1) $L \triangleq GCD(M_1, N_2)$

(2) $If\ (K_{R1} + K_{L2})\ MOD\ L \neq O$

- Delete path
- Go to "DONE"

(3) $\overline{M}_1 = \frac{M_1}{L};\ \ \overline{N}_2 = \frac{N_2}{L};$

(4) Use Euclid's algorithm to find P_1, P_2, such that

$$P_1\overline{M}_1 + P_2\overline{N}_2 = 1$$

(5) Define the equivalent values for the new path

$$K_{Leq} = K_{L1} + N_1 K_{M_1} + \frac{K_{R1} + K_{L2}}{L} P_1 N_1$$

$$K_{\mathrm{Re}q} = K_{R2} + M_2 K_{M_2} + \frac{K_{R1} + K_{L2}}{L} P_2 M_2$$

$$K_{Meq} = 0$$

$$N_{eq} = N_1 \overline{N}_2$$

$$M_{eq} = \overline{M}_1 M_2$$

(6) "DONE"

[0045] Returning now to FIG. 2, the present invention further determines whether an implementation assignment command has been received at step 64. Step 66 replaces an original component of the circuit with one or more implementation components based on the implementation assignment command. Specifically, FIGS.

7A and 7B illustrate that an original component 68 (although not necessarily a component of the behavioral description) has one or more implementation components 70 that are equivalent to the original component 68. The implementation components 70 are contained within the library of foundry primitives and have unambiguous HDL descriptions that are associated with them. Typically, the implementation components are adders and multipliers. For example, if implementation component 70a is a multiplier that multiplies two 5-bit numbers and produces a 10-bit result, this component can be readily represented in HDL.

[0046] Returning now to FIG. 2, the present invention determines whether a delay shifting command has been received at step 72. Step 74 moves a delay component from a first location in the circuit to a second location in the circuit based on the delay shifting command. Specifically, FIG. 8 illustrates the rearrangement of an original architecture 76 by moving a delay component 78 from a first location in the circuit to a second location. The resulting architecture 80 has a total delay that is maintained.

[0047] The hierarchical netlist has a unique file structure that is represented in several equivalent forms throughout the transformation process. An element in the netlist is represented by a N-tuple, defining:

Table 1

Tuple #	Parameter
1	element type
2	input ports
3	- from
4	output ports
5	- to
6	level in hierarchy

N	context dependent flags

[0048] The design tool architecture therefore accepts inputs at the behavioral level and automatically generates both compilable code at the HDL level and C-code for efficient system simulation. When the tool architecture is script driven, automatic transformation of the behavioral description (either block diagram or netlist) into a structural HDL form is possible. In the structural HDL form, realizable foundry library components (such as adders and multipliers) are substituted for idealized components and architecture sharing and bit precision are defined. The automated code generation and associated file structure produced by the design tool architecture allow the design time from system concept to delivery of compiled code for the resulting ASIC to be reduced by as much as a factor of 10.

[0049] The file structure allows for efficient 2-way travel through the hierarchical netlist when executing the transformations. Furthermore, the tool architecture allows for a systematic hierarchical proof of equivalence at every transformation, rather than requiring a much more complex (and often impossible) proof of overall equivalence at the completion of compilation. The command sets (i.e., transformations) are preferably script driven and are intended to be reusable to efficiently optimize a given problem architecture as well as to solve similar functional problems.

[0050] The design tool and method according to the present invention also inserts markers into a behavioral block diagram and associated netlist to confirm that when the netlist is processed or parsed into a set of symbolic equations, the resulting symbolic equations are finite and computationally feasible. The marker augmentation

disclosed herein is compatible with the existing design tool methodology that is disclosed in U.S. Patent No. 6,077,303, which is incorporated by reference. More specifically, the marker augmentation is compatible with theorems relating to the manipulation of delays and generalized rate change components. The design tool, according to the invention, generates symbolic equations with markers that are provably equivalent to the HDL code that defines the ASIC design. The symbolic equations with markers are then compared to reference or textbook equations to prove that the VHDL design is functionally equivalent to the original high-level behavioral block diagram. As part of the proof, the symbolic equations with markers are simplified in a symbolic manipulation program where each of the output variables is eventually traced back to the set of input values that determine it.

[0051] Referring now to FIG. 9, an exemplary circuit topology 10 is illustrated. The exemplary circuit topology 10 has a source component 12 that has a symbolic string output "x[1]". An output of the source component 12 is input to a unit delay component 14 that provides a unit delay (z^{-1}) and has an symbolic string output "x[0]". An output of the source component 12 is also input to an ideal multiplier component 16. An output of the unit delay component 14 is input to an ideal multiplier component 18. A constant component 20 has a symbolic string output "h[0]" that is equal to a constant number. A constant component 22 has a symbolic string output "h[1]" that is equal to a constant number. A symbolic string output of the ideal multiplier component 16 is equal to "x[1] * h[0]". A symbolic string output of the ideal multiplier component 18 is equal to "x[0] * h[1]". The outputs of the ideal multipliers 16 and 18 are input to an ideal adder component 26. A symbolic string output of the ideal adder component 26 is equal

to "x[1] * h[0] + x[0] * h[1]". The circuit topology 10 has a sink component 28 that is associated with a symbolic string "Z". A symbolic equation that is equal to "z := x[1] * h[0] + x[0] * h[1]" is printed to a symbolic equation file.

[0052] Referring now to FIGs. 10 and 11, the design tool is modified according to the invention to print the set of equations at each node of the block diagram on each clock cycle. For example, FIG. 10 illustrates time t = 1. In FIG. 10, reference numbers from FIG. 9 will be used to denote similar elements. The modifications resulted in a large number of relatively short equations being written into the equation file for subsequent comparison to the reference equations. In the circuit topology 10, the short length of the individual equations present at any clock cycle considerably eases the computational and storage burden on the HDL simulation program that is used to generate these equations from the symbolic HDL code. The HDL simulation program can be a ModelTech HDL simulator or any other suitable simulation program. As was true with FIG. 9, the symbolic manipulation program back-substitutes from one equation to the next until the output variables are expressed in terms of the set of input values that determined them.

[0053] The source component 12 of the circuit topology 10' has a string output of "x[1]". The output of the source component 12 is input to the unit delay component 14 that has a string output s[14,0,1] where the string output is defined as s[ID #, pin #, time]. The string output of each component must be a unique variable. The constant component 20 has a string output h[0]=1. The constant component 22 has a string output h[1]=2. String output of the ideal multiplier component 16 is equal to s[16,0,1]. A string output of the ideal multiplier component 18 is equal to s[18,0,1]. A

string output of the ideal adder component 26 is equal to s[26,0,1] = s[16,0,1] + s[18,0,1]. The string output z[1] := s[26,0,1] is printed to a file for time t = 1.

[0054] In some cases, however, such as a cascade of filter functions or the expression of closed loops, the resulting number of terms and equations grows quickly due to the nature of the circuit topology. For example, Fig. 12 illustrates an equation that grows explosively and fails to keep the resulting set of equations computationally reasonable. Fig. 12 includes a circuit topology 48 with a first component 50 having an input x and an output y and a second component 52 having an input y and an output z. The resulting set of equations for the circuit topology 48 grows quickly and is computationally difficult.

[0055] Referring now to Fig. 13, in order to keep the resulting set of equations computationally feasible, the design tool allows the designer to partition the circuit topology 48 with one or more marker components 54. Despite the insertion of the marker component 54, the design tool automatically transforms the netlist to produce both numeric and symbolic files for subsequent compilation to gates and proof of functionality. As a result of the insertion of the marker component 54 into the circuit topology 48 that describes a cascade of filters, the equation with n^2-order terms is transformed into two equations each having n-order terms and the computational requirements are reduced.

[0056] Referring now to Figs. 14 and 16, the high-level functional requirements of the marker component are summarized. The marker component is defined using a symbolic name. During HDL simulation, the marker component uses current time counts of each clock cycle to compute an index. The marker component computes a symbolic string for the output node in the HDL simulation. The form of the

symbolic string is marker_name[index]. Preferably, the marker_name is a unique prefix. For example, if the marker_name is equal to y, the input string is equal to x[n], and the output string is equal to y[n], the symbolic equation "mark y[n] = x[n]".

[0057] Referring now to Fig. 15, steps for handling a marker component are illustrated. Control begins at step 100. In step 102, control keeps a running count of time during the simulation. In step 104, the index is computed. In step 106, the output string is set equal to "y[%d]", index". In step 108, control prints ["mark y[%d=%s", index, INstring] to the equation file. Control ends at step 110.

[0058] Referring now to Figs. 17 and 18, a marker component 60 is inserted into the example illustrated in Fig. 10. Reference numbers from FIG. 10 will be used where appropriate to identify similar components. The circuit topology 10" has a source component 12 that has a symbolic string output "x[1]". The output of the source component 12 is input to the unit delay component 14 that has a modified symbolic string output s[14,0,1]. The constant component 20 has a modified symbolic string output h[0]=1. The constant component 22 has a modified symbolic string output h[1]=2. A modified symbolic string output of the ideal multiplier component 16 is equal to s[16,0,1] and is input to the marker component 60. An output of the marker component 60 is input to the ideal adder component. A modified symbolic string output of the ideal multiplier component 18 is equal to s[18,0,1]. An output of the marker component 60 "mark y[1]:= s[16,0,1] is input to the ideal adder component. A modified symbolic string output of the ideal adder component 26 is equal to s[26,0,1] = y[n] + s[18,0,1]. The block diagram 10 has the sink component 28 "z[1]". The modified symbolic equation equal to z[1] := s[26,0,1] is printed to a modified symbolic equation file for time t = 1.

[0059] As was previously discussed, the output of each component must have a unique variable name. FIG. 19 illustrates a hierarchy with an exemplary component output variable naming convention. The hierarchy includes a top entity 100 with three instantiations of entity A (104,106, and 108) into the top entity 100, two instantiations of entity B (110 and 112) into the entity A 104, and the instantiation of a multiplier 114 into the entity B 110. Two instantiations of entity B (118 and 120), are created in the entity A 106. Two instantiations of entity B (122 and 124) are created in the entity B 108.

[0060] One suitable naming convention to define unique variable names where s[hierarchy path list, pin #, count] is either a time stamp or an event stamp. A trace of a hierarchy path defines a unique ID. For example, the entity A 104 has a unique component ID s[104,O,n] and the entity B 110 has a unique component ID s[104,110,09,n]. The multiplier has a unique component ID s[104,110,114,0,n]. Skilled artisans can appreciate that other suitable naming conventions can be employed. For example, the unique component ID can be s[104.110.114,0,n], s[104-110-114,O,n], etc. FIG. 20 illustrates internal stored representations of the hierarchy of FIG. 19.

[0061] Referring now to FIG. 21 and the derivation below, a detailed description of the marker index is defined in terms of a generalized rate change component. The resulting index is required to make the partitioned equations recognizable, even after netlist transformations and element sharing. The index derivation is fully generalized:

$$a[n]=x[n+K_L]$$

$b[n]=a[Nn]$

$c[n]=b[n+K_M]$

$$d[n]=\begin{cases} c\left[\frac{n}{M}\right] & \text{nmcd } M=0 \\ 0 & \text{else} \end{cases}$$

$y[n]=d[n+K_R]$

Substitute back for y[n]

$$y[n]=\begin{cases} c\left[\frac{n+K_R}{M}\right] & (n+K_R) \text{ Mod } M=0 \\ 0 & \text{else} \end{cases}$$

$$c\left[\frac{n+K_R}{M}\right]=b\left[\frac{n+K_R}{M}+K_m\right]$$
$$=a\left[\left(\frac{n+K_R}{M}+K_M\right)N\right]$$
$$=x\left[\left(\frac{n+K_R}{M}+K_M\right)N+K_L\right]$$

Thus,

$$y[n]=\begin{cases} x\left[\left(\frac{n+K_R}{M}+K_m\right)N+K_L\right] & (n+K_R)\,\text{Mod}=c \\ 0 & \text{else} \end{cases}$$

$$\text{At time} = \left(\left(\frac{n + K_R}{M}\right) + K_M\right) N + K_L \text{ set index} + n$$

$$\text{Thus } t = \left(\frac{i + K_R}{M} + K_M\right) N + K_L$$

write index (i) as fn of time (t)

$$i = (t - K_L)\frac{M}{N} - (K_M + K_R) \quad (t - K_L) \text{ Mod } N = 0$$

[0062] Those skilled in the art can now appreciate from the foregoing description that the broad teachings of the present invention can be implemented in a variety of forms. Therefore, while this invention can be described in connection with particular examples thereof, the true scope of the invention should not be so limited since other modifications will become apparent to the skilled practitioner upon a study of the drawings, specification and following claims.